Cloud-computing for non-interactive long-running computationally intensive scientific applications

Lapin, Andrei ; Schiller, Eryk ; Kropf, Peter

Abstract: Often, there is a particular type of applications in different scientific domains, i.e., non-interactive long-running computationally intensive applications. Cloud-computing looks as a perfect environment for executing this type of applications. In this paper, we present a system, which exploits advantages of the cloud-computing environment and offers researchers an easy-way to execute the chosen type of applications "in the clouds". Also, we show performance comparison of an example application executed with our system and on the bare metal hardware.

Originally published at:
Lapin, Andrei; Schiller, Eryk; Kropf, Peter (2015). Cloud-computing for non-interactive long-running computationally intensive scientific applications. In: Autonomous Systems (AutSys2015), Cala Millor, Majorca, Spain, 25 October 2015 - 30 October 2015. VDI Verlag, 221-231.

Cloud-computing for non-interactive long-running computationally intensive scientific applications

Andrei Lapin[1], Eryk Schiller[2] and Peter Kropf[1]

[1]University of Neuchatel, Switzerland

[2]University of Bern, Switzerland

Abstract: Often, there is a particular type of applications in different scientific domains, i.e. non-interactive long-running computationally intensive applications. Cloud-computing looks as a perfect environment for executing this type of applications. In this paper, we present a system, which exploits advantages of the cloud-computing environment and offers researchers an easy-way to execute the chosen type of applications "in the clouds". Also, we show performance comparison of an example application executed with our system and on the bare metal hardware.

1 Introduction

Often, there are local High-Performance Computing (HPC) infrastructures (e.g., clusters, grids) maintained by universities or research centers. At some point, no matter how powerful the infrastructure is, scientists may reach the limit of the available resources and experience difficulties to accommodate more data or jobs. At this point, a possible solution may be to use remote resources as an extension for the locally available resources. Simple application migration to a different infrastructure is usually impossible because these infrastructures may significantly differ and may be incompatible in terms of application migration. Such incompatibility may be caused by differences in hardware architecture or by the installed software (e.g., incompatible versions). Moreover, scientific applications are often dependency-rich and may rely on many libraries, which makes it extremely difficult to reproduce the working environment on another machine. Described issues often make a scientific application bounded to a single platform and the migration process a complicated operation. Researches obliged to stay with technologically similar HPC infrastructures and use portable programming solutions despite its efficiency for their research, or they must design infrastructure-oriented thereby nearly unportable applications.

2 Cloud-Computing for science

2.1 Virtualization

Virtualization is a very old concept that dates back to 1974 when Popek and Goldberg derive the first set of formal requirements for efficient virtualization [4]. The idea is to run several instances of the Operating System (OS) called guests or Virtual Machines (VMs) on a single physical machine (bare metal) hereafter called the host.

Virtualization is provided through a special piece of software called a hypervisor or VMM running on a physical machine. There are in general two types of hypervisors defined in the literature: hypervisor type I and hypervisor type II. Hypervisor type I is a micro-kernel running on top of a bare metal that performs tasks such as VM scheduling, resource management, and access to peripherals. Contrary to this, the hypervisor of type II runs guests on top of the host OS; therefore, most of the tasks performed by the hypervisor might be delegated to the host OS allowing for a lightweight design of the VMM. Top-level architectural differences between hypervisors are depicted in Figure 1.

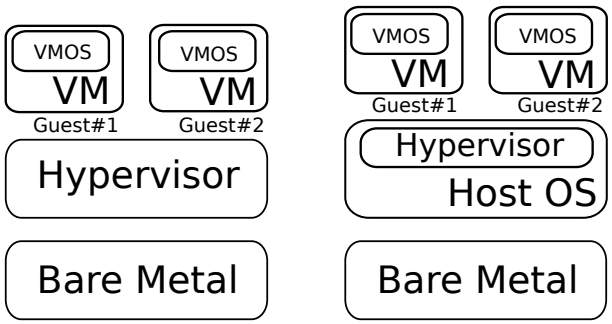

Fig. 1: Top-level architectural difference between Hypervisors I (left), II (right).

In our research, we mainly consider and study two open-source and nowadays the most popular hypervisors of type 2, i.e. XEN [5] and Kernel-based Virtual Machine (KVM) [2], which both show performance reduction of the guests of around 5% in comparison to the host [6].

2.2 Cloud-computing

A reasonable performance drop of around a few percent for a guest OS was a key to establish a service that provides computing resources on-demand in terms of VMs. Cloud computing is a collection of computing technologies targeted to provide computing as a service. For the end-user cloud computing can be an easy-to-use technology and at the same time suitable for complex system deployments. The main advantages for the end-user are characterized by the following features of cloud computing [3]:

- **Rapid elasticity** and **scalability** via dynamic resource provisioning on self-service basis in near real-time.
- **Measured service**, cloud computing resource usage can be measured, controlled, and reported transparently for both the provider and consumer.
- **On-demand self service**, the consumer can acquire computer services such as applications, networking or server services without requiring interactions with each service provider.
- **Broad network access** to the cloud infrastructure is available over the Internet and through standard mechanisms and protocols.
- **Resource pooling**, computing resources are pooled together to serve multiple consumers using multiple-tenant model. Resources can be dynamically assigned and reassigned according to consumer demand.

Cloud computing comes with three main models of resource provisioning. These models differ in type of resources they offer:

- **The Software as a Service (SaaS) model** delivers access to software applications over the web. The applications are centrally managed on the cloud possibly by a third-party vendor and can be accessed with a web browser without any preliminary installation. This approach simplifies the application usage experience for the end-users by excluding necessity to update the software or deal with licenses.
- **The Platform as a Service (PaaS) model** delivers an application development platform allowing developers to concentrate on software design, development itself and deployment rather than maintenance of the underlying hardware.

- **The Infrastructure as a Service (Iaas) model** delivers any kind of system resources on demand (e.g., compute power, storage, networking) and allows users to pay for it based on consumption.

Virtualization software allows to separate physical infrastructure and create dedicated virtual resources. Virtualization is a key component of the Infrastructure as a Service cloud, which enables tenants to receive access to computing resources through instantiated VMs. The tenant can scale resources in or out to meet his/her demands, for example, by releasing VMs on idle, or requesting new VMs upon a work-load burst. However, cloud computing is not virtualization. Cloud computing is a set of technologies including virtualization which are combined together to offer computing as a service and specifies how resources are available (IaaS, Paas, SaaS). Virtualization is not necessary offered as a service.

2.3 Motivation, Challenges and Requirements

Cloud computing evolves extremely quickly and offers a promising solution to the problems described in the section 1 due to its scalability and on-demand self service features. However, giving researchers access to a cloud platform is not enough. Designing an efficient distributed cloud-based application requires certain knowledge, and even with that knowledge, the application development process would become complicated for a someone who is not a computer scientist.

Our research is motivated by the need to design a system that does not face the previously mentioned problems and allows researchers to benefit from the advantages of cloud infrastructure and use it as a typical HPC infrastructure. We concentrate on the following aspects of our system:

Service-Oriented design. System has to fulfill the service-oriented concept, meaning that the user does not require to maintain hardware infrastructure for hosting the system and can only rent remote virtualized resources from a cloud provider.

Portability. Simple and quick deployment and redeployment of the system on various platforms.

Heterogeneous resource utilization. Ability to combine different types of local and remote resources and use them together in a single resource pool.

Resource selection and resource necessity prediction. Appropriate resource selection mechanism for a specific application. Minimization of the VM initialization time by predicting VM necessity and preliminary VM spawning mechanisms.

Simple context replacement. Ability to add various scientific applications which the system can execute with performing no changes in the system core.

Additionally, we need to make sure that cloud infrastructure could be efficiently used for a common type of scientific application, i.e. non-interactive long-running computationally intensive applications.

3 CLAUDE: Cloud-Based Computing-Oriented Service

This section introduces our CLAUDE system. In the following, we present architecture of the system and an example application which we used to measure the system performance.

3.1 System architecture

The system consist of three main components, i.e. *the Communication Interface*, *the Application Controller* and *the Centralized Storage*. Figure 2 represents the CLAUDE system architecture and inter-component information flows. *The Communication Interface* and *the Application Controller* have to be installed manually by the user. There are no specific requirements for the installation. It can be a local or remote installation on a single or separate machines. The user is allowed to make this decision according to his/her requirements and available resources. The third component (i.e. *the Centralized Storage*) is a Cloud-Based Storage service which is often offered by cloud providers. In this case, the user does not need to deploy his/her own *Centralized Storage* and may use a remote solution of a cloud provider. *The Communication Interface* and *the Application Controller* interact through message exchanges. We use the standardized AMQP protocol and the RabbitMQ messaging system on both sides. We defined certain types of messages to control applications execution life-cycle, most of the messages work in the request/response manner. Each component knows what types of messages it can send and receive thus contains appropriate logic for processing the corresponding messages. In *the Centralized Storage*, we keep input/output data of the applications executed through the system. The user can access the data through *the Communication Interface*. We use the standardized S3 protocol to interact with the storage thus, it is required for *the Centralized Storage*

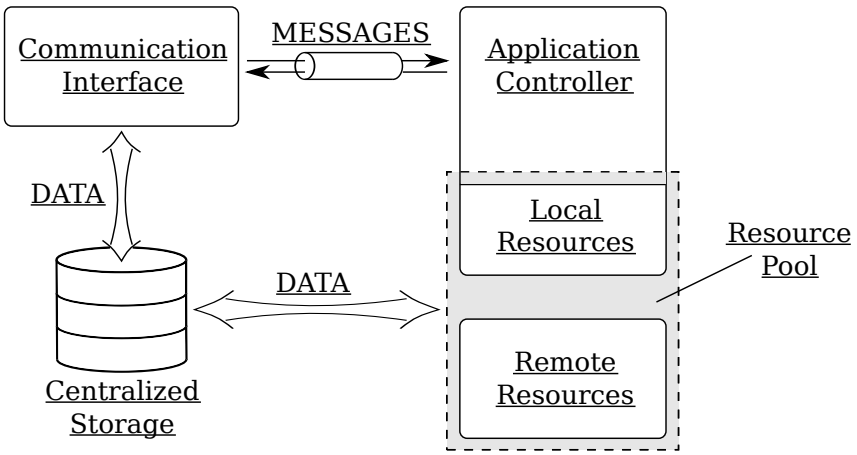

Fig. 2: The CLAUDE system architecture

to support this protocol. For distributing and executing applications, *the Application Controller* uses predefined resources from *the Resource Pool*. To define available resources the user has to describe it (i.e. RAM, CPU) and provide an interface through which *the Application Controller* can access the resource. *The Resource Pool* combines available local and remote resources, and *the Application Controller* chooses an appropriate resource combination for a particular application. We designed the system in a way that components remain loosely coupled and communicate only through standard protocols. The user always communicates with the system through *the Communication Interface*. One of the most convenient and intuitive implementations of *the Communication Interface* might be a Web-based application which may allow users to have additional meta-data information concerning applications execution state or input/output data.

The core component of the system is *the Application Controller*. Figure 3 shows main subcomponents of *the Application Controller*. *The Controller* takes care of the applications scheduling and maintaining correct relations between running applications and corresponding computational resources. *The Session Storage* is used for keeping backups of existing relations that the system could recover in case of machine failure. Last three subcomponents, i.e. *the Statistics*, *the Resource Selection* and *the Resource Prediction* are mainly responsible for improving the overall system performance by preliminary VM spawning and predicting what resource combination is necessary for a particular application.

3.2 Example application

To test our system we have chosen a widely used hydro-geological application *HydroGeoSphere (HGS)* which perfectly matches the initial requirements, i.e. it

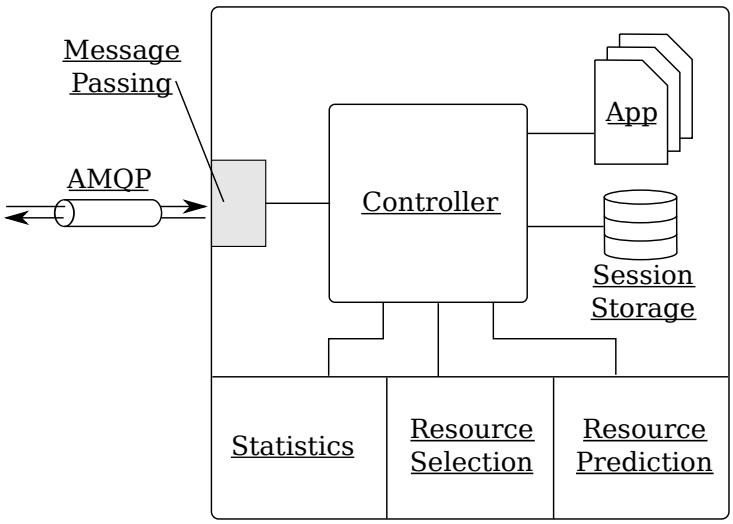

Fig. 3: Application controller

is a non-interactive long-running computationally intensive application. The application has been designed to solve a variety of hydro-geological problems with different complexity (e.g., regular geometry, steady-state saturated flow, integrated flow, solute, heat transport in regular-scale catchments). *The HGS* performs physically based simulations and uses numerical methods to solve nonlinear equations [1]. The execution time may vary from minutes to hours depending on the corresponding input and the problem complexity.

Figure 4 represents *the HGS* application life-cycle in *the CLAUDE* system. The

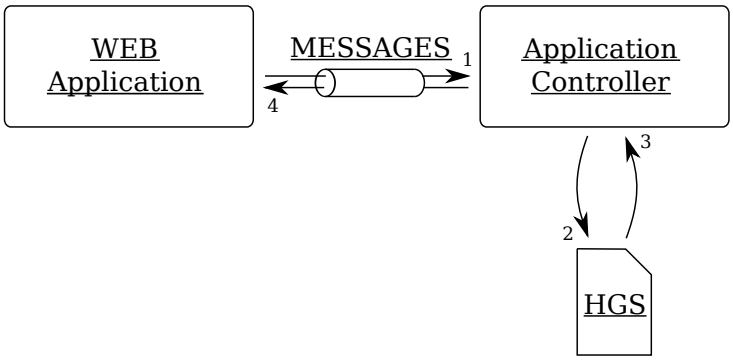

Fig. 4: HydroGeoSphere application

life-cycle begins when the user submits an *HGS* input to *the CLAUDE* system

through *the Web Interface*. After the corresponding upload completes, an appropriate message is transmitted to *the Application Controller* containing various application-related information (e.g. application type, input location, optional auxiliary user-defined characterization of the expected computation complexity). When *the Application Controller* receives the message, it predicts amount of resources required for the application based on previously built application-specific resources consumption model and auxiliary user-defined information. Next step is to spawn an appropriate VM and start the application execution on the VM. After the execution is completed, *the Application Controller* retrieves the execution-related information and sends an appropriate message back to *the Web Interface*. When *the Web Interface* receives the message the life-cycle ends and the user is allowed to download application output.

4 Evaluation

This section evaluates the performance of the chosen *HGS* application in various environments and shows the performance drop which comes with virtualization. We focused on the most important evaluation criteria, which is in this scenario the application execution time. For the experiments, we used four different HGS input models (i.e. model_1, model_2, model_3, model_4) with different complexity of the computations. Each experiment we executed multiple times to calculate the average execution time and the standard deviation.

The HGS application performs in two steps: the first step includes most of the I/O operations, data preparation and memory allocation; and the second step executes actual hydro-geological problem solving. From the OS point of view, these two steps are significantly different. The first step requires many instructions to be executed in the OS privileged mode (the Kernel mode) unlike the second step, which substantially requires user mode only. We have performed two sets of experiments, where each set was targeted to show the performance of a particular step.

In our first set of experiments, we measured the execution time of the computationally intensive step (the second step). Indeed, a typical computationally intensive application in the virtual environment performs very close (in terms of execution time) to the bare metal execution (Figure 5). This is the result we expected to see because most of the instructions are executed in user mode directly on the CPU without any interaction with the hypervisor and host OS.

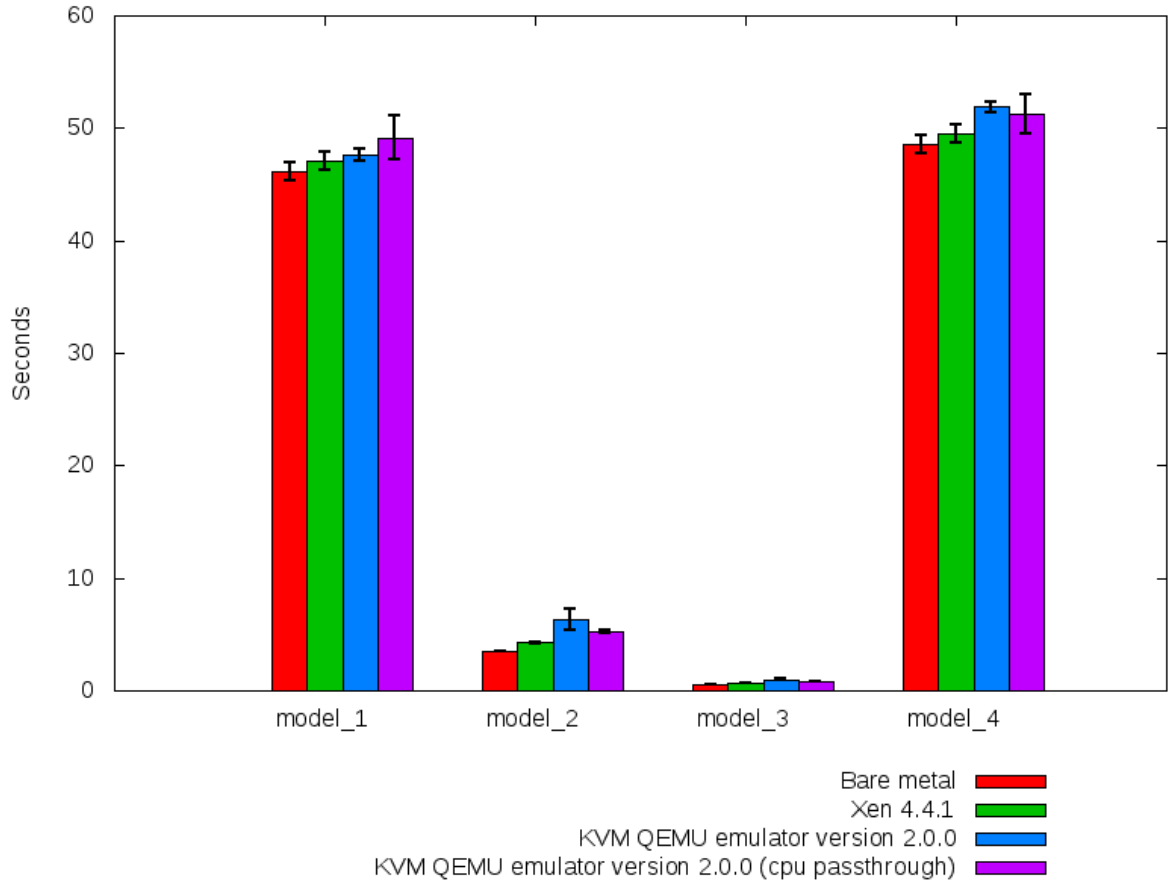

Fig. 5: Time spent in user mode

In our next set of experiments, we measured the execution time for the kernel mode instructions (the first step matches pretty well this requirement). The kernel mode instructions cannot be directly executed on the CPU and first have to be passed to the hypervisor and the host OS (e.g., I/O operations, memory allocation) which certainly increases the execution time. Figure 6 shows the execution time comparison in different environments. We see that the performance drop is more significant, however, it does not severely reduce the overall execution performance because for our application the majority of time is always spent for the computationally intensive step (see Figure 5).

5 Conclusion

Cloud computing is a very promising technology for non-interactive computationally intensive scientific applications. It gives performance very close to the bare metal and allows the user to have a required number of simultaneously running application instances and pay only for the consumed resources. Addi-

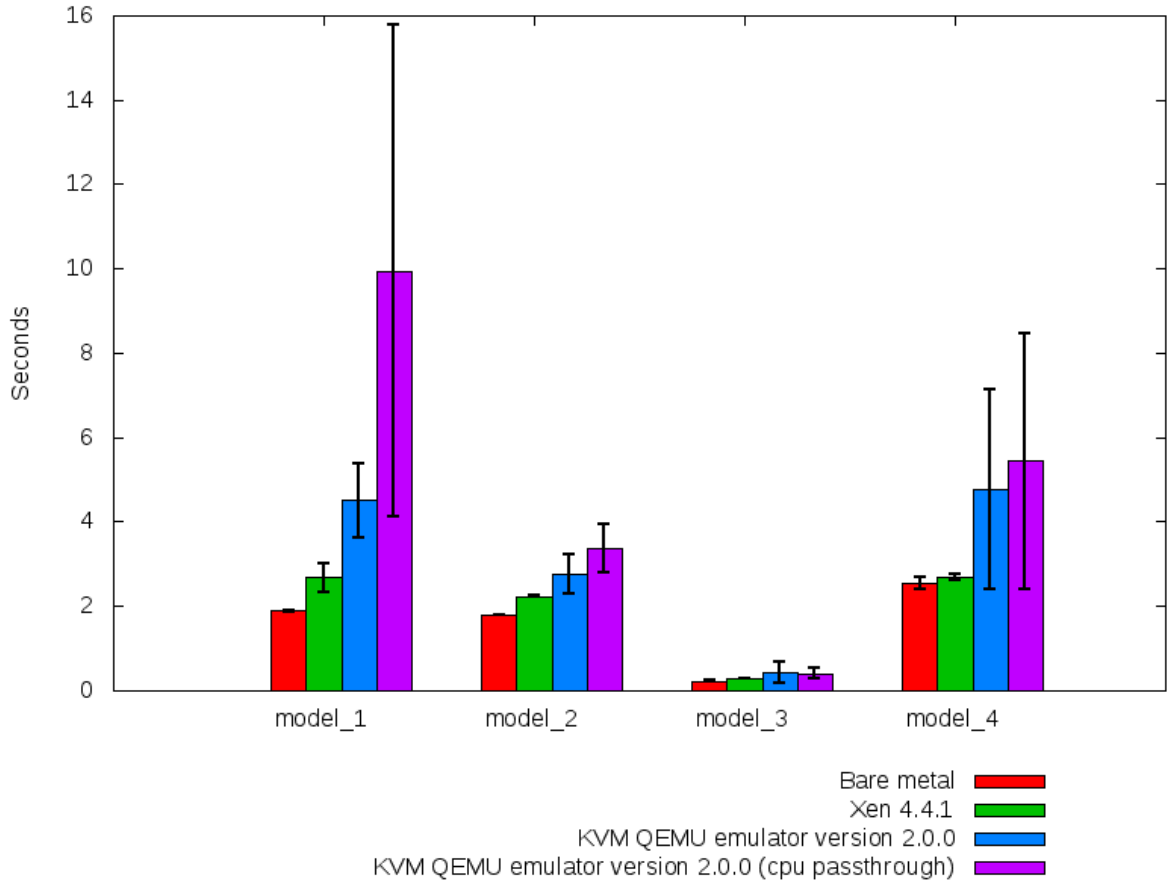

Fig. 6: Time spent in the kernel mode

tionally, it releases the user from necessity to maintain and house any physical infrastructure.

We have designed and implemented a system which allows researchers from different domains to efficiently use cloud-computing resources for non-interactive long-running computationally intensive applications without necessity to deal with computer science related questions and problems.

We have tested the system and showed that the virtual environment performance is very close to the native bare metal execution performance for the chosen type of applications. However, there is still a room for future research and improvements.

Bibliography

[1] Philip Brunner and Craig T. Simmons. "HydroGeoSphere: A Fully Integrated, Physically Based Hydrological Model". In: *Ground Water* 50.2 (2012), pp. 170–176. ISSN: 1745-6584. DOI: 10.1111/j.1745-6584.2011.00882.x. URL: http://dx.doi.org/10.1111/j.1745-6584.2011.00882.x.

[2] KVM. *http://www.linux-kvm.org/*. URL: http://www.linux-kvm.org/.

[3] Peter M. Mell and Timothy Grance. *SP 800-145. The NIST Definition of Cloud Computing*. Tech. rep. Gaithersburg, MD, United States, 2011.

[4] Gerald J. Popek and Robert P. Goldberg. "Formal Requirements for Virtualizable Third Generation Architectures". In: *Commun. ACM* 17.7 (July 1974), pp. 412–421. ISSN: 0001-0782. DOI: 10.1145/361011.361073. URL: http://doi.acm.org/10.1145/361011.361073.

[5] The Xen Project. *http://www.xenproject.org/*. URL: http://www.xenproject.org/.

[6] S. Varrette et al. "HPC Performance and Energy-Efficiency of Xen, KVM and VMware Hypervisors". In: *Computer Architecture and High Performance Computing (SBAC-PAD), 2013 25th International Symposium on*. 2013, pp. 89–96. DOI: 10.1109/SBAC-PAD.2013.18.